.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 1
Introducción y primeros años

2 de diciembre de 2024

Damaris Juares
108566

Lucas Perez Esnaola
107990

Índice

1. Introducción

1.1. Contexto

Cuando Mateo nació, Sophia estaba muy contenta. Finalmente tendría un hermano con quien jugar. Sophi tenía 3 años cuando Mateo nació. Ya desde muy chicos, ella jugaba mucho con su hermano.

Pasaron los años, y fueron cambiando los juegos. Cuando Mateo cumplió 4 años, el padre de ambos le explicó un juego a Sophia: Se dispone una fila de n n monedas, de diferentes valores. En cada turno, un jugador debe elegir alguna moneda. Pero no puede elegir cualquiera: sólo puede elegir o bien la primera de la fila, o bien la última. Al elegirla, la remueve de la fila, y le toca luego al otro jugador, quien debe elegir otra moneda siguiendo la misma regla. Siguen agarrando monedas hasta que no quede ninguna. Quien gane será quien tenga el mayor valor acumulado (por sumatoria).

El problema es que Mateo es aún pequeño para entender cómo funciona esto, por lo que Sophia debe elegir las monedas por él. Digamos, Mateo está "jugando". Aquí surge otro problema: Sophia es muy competitiva. Será buena hermana, pero no se va a dejar ganar (consideremos que tiene 7 nada más). Todo lo contrario. En la primaria aprendió algunas cosas sobre algoritmos greedy, y lo va a aplicar.

1.2. Consigna

1. Hacer un análisis del problema, y proponer un algoritmo greedy que obtenga la solución óptima al problema planteado: Dados los n valores de todas las monedas, indicar qué monedas debe ir eligiendo Sophia para si misma y para Mateo, de tal forma que se asegure de ganar siempre. Considerar que Sophia siempre comienza (para sí misma).

2. Demostrar que el algoritmo planteado obtiene siempre la solución óptima (desestimando el caso de una cantidad par de monedas de mismo valor, en cuyo caso siempre sería empate más allá de la estrategia de Sophia).

3. Escribir el algoritmo planteado. Describir y justificar la complejidad de dicho algoritmo. Analizar si (y cómo) afecta la variabilidad de los valores de las diferentes monedas a los tiempos del algoritmo planteado.

4. Analizar si (y cómo) afecta la variabilidad de los valores de las diferentes monedas a la optimalidad del algoritmo planteado.

5. Realizar ejemplos de ejecución para encontrar soluciones y corroborar lo encontrado. Adicionalmente, el curso proveerá con algunos casos particulares que deben cumplirse su optimalidad también.

6. Hacer mediciones de tiempos para corroborar la complejidad teórica indicada. Agregar los casos de prueba necesarios para dicha corroboración. Esta corroboración empírica debe realizarse confeccionando gráficos correspondientes, y utilizando la técnica de cuadrados mínimos. Para esto, proveemos una explicación detallada, en conjunto de ejemplos.

7. Agregar cualquier conclusión que les parezca relevante.

2. Análisis del problema

2.1. Descripción de problema

El problema para el Juego de Hermanos en su version codiciosa es usar una regla simple para seleccionar las monedas de Sofia y las de Mateo, esta regla tiene que poder llevarnos a que Sofia siempre gane las partidas acumulando una suma mayor a la de su hermano.

2.2. Propuesta de algoritmo greedy

Lo primero que necesitamos es elegir una norma de eleccion que cumpla con nuestro objetivo: que Sofia gane.

A continuación mostraremos el análisis de las diferentes alternativas que propusimos, y como llegamos a la alternativa elegida:

2.2.1. Primera propuesta

Como primera idea para la regla greedy se nos ocurrió elegir la moneda de mayor valor (entre las disponibles segun las reglas del juego) en cada turno de Sofia y en el turno de Mateo utilizar el mismo criterio. Para este caso, si bien siempre empieza jugando Sofia y eso le da una ventaja, tener el mismo criterio para ambos jugadores no le garantiza a nadie la victoria. Si tomamos como ejemplo a las monedas [6,8,1,2,4,5], siguiendo nuestra propuesta de regla, las elecciones para cada jugador se verían de la siguiente manera:

Elecciones SOFIA: 6, 5, 2. Suma: 13

Elecciones MATEO: 8, 4, 1. Suma: 13

Lo cual no nos lleva a la solución buscada.

2.2.2. Segunda propuesta

Como segunda alternativa, propusimos elegir la moneda de mayor valor en cada turno de Sofia y en el turno de Mateo elegir la moneda rechazada por Sofia.

Elecciones SOFIA: 6, 8, 2. Suma: 16

Elecciones MATEO: 5, 4, 1. Suma: 11

En este caso nuestra eleccion cumple con nuestros requisitos. Sin embargo, no siempre devuelve la solución óptima.

En este otro caso, con monedas [6,2,1,2,4,5], las elecciones segun nuestra regla greedy serian:

Elecciones SOFIA: 6, 4, 2. Suma: 12

Elecciones MATEO: 5, 2, 1. Suma: 8

Cuando la solución óptima es:

Elecciones SOFIA: 6, 5, 4. Suma: 15

Elecciones MATEO: 2, 1, 2. Suma: 5

2.2.3. Propuesta final

Nuestra propuesta final como forma de tomar nuestra decision sobre cada turno es mantener el criterio de seleccion anterior para Sofia pero con la modificacion de elegir la menor moneda de menor valor entre las restantes para Mateo.

Para el primer ejemplo dado, con monedas [6,8,1,2,4,5], los jugadores harían las siguientes elecciones:

Elecciones SOFIA: 6, 8, 4. Suma: 18

Elecciones MATEO: 5, 1, 2. Suma: 8

Esta regla nos lleva a nuestro objetivo de que gane Sofía, además de dar la solución óptima.

Supondremos que Sofia es muy codiciosa y elegiremos la ultima opcion descripta para asegurale la mayor suma posible ademas de ganar.

2.2.4. Pseudocódigo

Algoritmo Greedy:

1. **Entrada:** Una lista de valores de las monedas.
2. **Proceso:** En cada turno:
   - Sophia elige la moneda de mayor valor entre la primera y la última de la fila.
   - Luego, Mateo elige la siguiente moneda de menor valor entre las dos primera y ultima opciones restantes.
3. **Repetir** hasta que no haya más monedas.
4. **Salida:** La lista de monedas que Sophia elige y la lista de monedas que Mateo elige, junto con la suma de las monedas elegidas por ambos.

Este enfoque asegura que Sophia siempre tomará la mejor opción posible en su turno, aprovechando el principio greedy de tomar la decisión óptima local en cada paso.

3. Demostración de optimalidad del algoritmo

3.1. Teorema de la solución óptima

El algoritmo greedy propuesto es óptimo porque, en cada turno, Sophia toma la moneda que maximiza su ganancia local. No se necesitan decisiones complejas que impliquen mirar más allá de un turno. Cada vez que toma la moneda de mayor valor, minimiza las opciones favorables para Mateo en su turno. Esto es suficiente para garantizar que Sophia obtenga el máximo valor posible.

3.2. Justificación

Como Sophia toma las mejores monedas para sí misma en cada turno, el algoritmo nunca deja de maximizar su ganancia, lo que lleva a una solución óptima. Por el otro lado, Mateo siempre toma las peores monedas, por lo que siempre esta minimizando su ganancia. De esta forma, siempre se llega a un resultado óptimo.

4. Algoritmo propuesto

A continuación, mostramos el código para el algoritmo greedy propuesto:

4.1. Código

```
def greedy(monedas):
    n= len(monedas)
    inicio= 0
    final=n-1

```

```
    acciones = [[], []]

    turno = 0 # 0: Sophia, 1: Mateo

    while n > 0:

        if turno == 0:
            if monedas[inicio] > monedas[final]:
                acciones[0].append(monedas[inicio])
                inicio+=1
            else:
                acciones[0].append(monedas[final])
                final-=1

        else:
            if monedas[inicio] > monedas[final]:
                acciones[1].append(monedas[final])
                final-=1
            else:
                acciones[1].append(monedas[inicio])
                inicio+=1
        n-=1

        turno = 1 - turno

    print(f"Ganancia Sofia: {sum(acciones[0])} - Monedas: {acciones[0]} \n")
    print(f"Ganancia Mateo: {sum(acciones[1])} - Monedas: {acciones[1]}")
```

4.2. Análisis de la complejidad

- El bucle principal es O(n), ya que procesa cada moneda una vez.
- El cálculo de las sumas al final es O(n).
- No hay operaciones costosas adicionales fuera del bucle.
- Se almacenan listas de acciones.

Por lo tanto, la complejidad temporal total es O(n) y la complejidad espacial es O(n).

4.3. Impacto de la variabilidad de valores a la complejidad temporal

La variabilidad de los valores de las monedas no afecta la complejidad temporal, ya que el algoritmo sigue ejecutándose en O(n), independientemente de los valores de las monedas. Esto se debe a que el algoritmo simplemente analiza si un valor es mayor que el otro, y esta operación no se ve influenciada por el número que esté comparando.

4.4. Impacto de la variabilidad de valores a la optimalidad

Los valores de las monedas afectan directamente a las decisiones de Sophia, ya que siempre elige la moneda de mayor valor disponible.

Sin embargo, la variabilidad de los valores no afecta la optimalidad del algoritmo, ya que el algoritmo sigue el principio de maximizar el valor de Sophia en cada turno, lo que garantiza el mejor resultado posible para ella.

5. Mediciones de tiempo

Para corroborar la complejidad teórica indicada (O(n)), se medirán los tiempos de ejecución del algoritmo greedy planteado con sets de datos generados aleatoriamente con distintos tamaños.

5.1. Generador de sets aleatorios

Para generar los sets aleatorios, se utilizó la siguiente función:

```
# Genera n sets de datos de tama o incremental para el problema de las monedas.
# Si se especifica una semilla, se utilizara para generar los datos.
def generar_set_datos_monedas(semilla, n):
    if semilla != None:
        seed(semilla)

    set_datos = []

    for i in range(1, n+1):
        set_datos.append([randint(1, 10000) for _ in range(i * 10)])

    return set_datos
```

De esta forma, se generan n sets de datos que van incrementando su tamaño iterativamente.

5.2. Mediciones

Generamos 100 sets de datos, donde cada set incrementa en 10 su tamaño frente al anterior. Las mediciones temporales al ejecutar el algoritmo greedy se muestran a continuación:

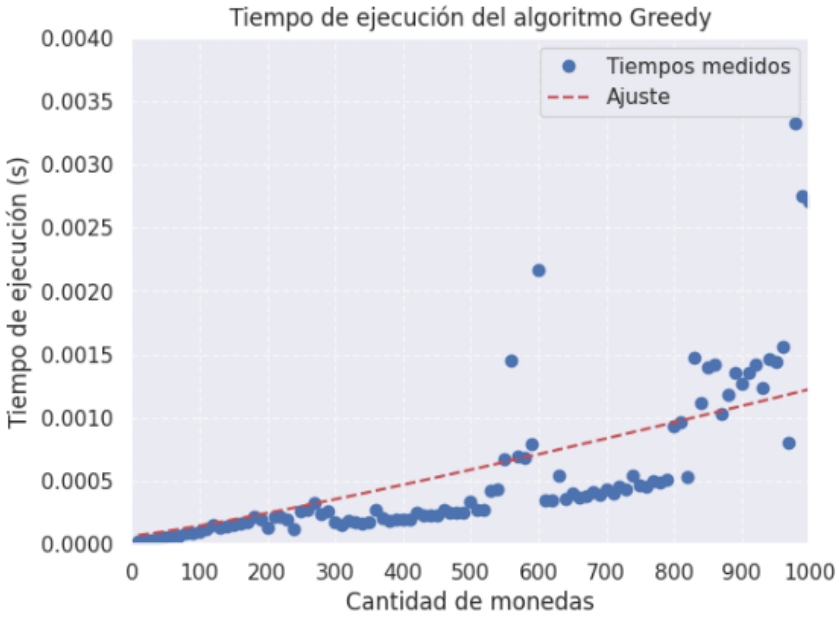

Figura 1: Mediciones temporales para el algoritmo Greedy

Para corroborar la complejidad teórica, se utilizó la técnica de cuadrados mínimos para ajustar los datos medidos a una recta lineal. Se ve claramente como para los primeros sets de datos la recta ajusta casi perfectamente, pero a medida que se incrementa la cantidad de monedas, el ajuste se desvía un poco. Igualmente, en términos generales podemos afirmar que la complejidad teórica indicada es correcta.

6. Conclusiones

En este primer trabajo práctico se analizó la tecninca Greedy para resolver el problema de las monedas.

Podemos afirmar que:

- El algoritmo greedy propuesto es eficiente y siempre proporciona la solución óptima para Sophia, dado que maximiza su valor en cada turno.
- La complejidad temporal es O(n), lo que lo hace adecuado incluso para listas grandes de monedas.
- La variabilidad de los valores no afecta la eficiencia, pero sí las decisiones de Sophia, lo que puede cambiar la distribución de las monedas elegidas por cada jugador.
- Las mediciones empíricas confirmarán la eficiencia del algoritmo y su comportamiento en diferentes tamaños de entrada.

Este análisis asegura que el algoritmo es eficiente, efectivo y óptimo para el problema planteado.